

One-loop integrals with XLOOPS-GiNaC*

C. Bauer, H.S. Do

Institut für Physik, Johannes-Gutenberg-Universität,
Staudinger Weg 7, D-55099 Mainz, Germany

Abstract

We present a new algorithm for the reduction of one-loop tensor Feynman integrals within the framework of the XLOOPS project, covering both mathematical and programming aspects. The new algorithm supplies a clean way to reduce the one-loop one-, two- and three-point Feynman integrals with arbitrary tensor rank and powers of the propagators to a basis of simple integrals. We also present a new method of coding XLOOPS in C++ using the GiNaC library.

*Supported by the Deutsche Forschungsgemeinschaft and the "Graduiertenkolleg Eichtheorien" at the University of Mainz.

1 Introduction

The purpose of this paper is twofold. First, we introduce a new method to code XLOOPS in C++ with the help of the GiNaC library [1]. This also serves as an introduction to GiNaC for symbolic computations.

Second, we introduce a new procedure for tensor reduction which allows us to manipulate one-loop one-, two- and three-point Feynman integrals with arbitrary tensor rank and arbitrary powers of propagators. Working in orthogonal and parallel space of momentum configuration [2], this procedure allows one to reduce the one-loop tensor integrals to a basis of simpler integrals without using the Passarino-Veltman procedure. This procedure has been implemented in the new version of the XLOOPS program, called XLOOPS-GiNaC.

Unlike the original XLOOPS, XLOOPS-GiNaC currently only handles the one-loop case (and as such competes with existing programs for one-loop calculations such as those presented in [7]) but it has been designed with being a prerequisite for two-loop problems in mind, and the authors' goal is to make XLOOPS-GiNaC a powerful tool for one- and two-loop analytical Feynman integrations.

In sections 2 and 3 we introduce briefly the motivation to rewrite XLOOPS in C++, and the GiNaC library. In section 4 the procedure used to reduce one-loop two- and three-point integrals is presented. In two appendices simple examples of C++ programs using the GiNaC library and the XLOOPS-GiNaC interface for one-loop integrals are presented.

2 XLOOPS-GiNaC: The Motivation

In the past, the XLOOPS package [6] has been developed in a heterogenous environment: The core routines for transforming Feynman graphs into the basic integrals and for analytic integration are implemented in the language of the **Maple** [8] computer algebra system, the graphical user interface is written in Tcl/Tk [9], and the numerical integration is done by C programs that are generated and compiled at run-time by XLOOPS.

This way of implementation has a couple of drawbacks:

- While **Maple** and other computer algebra systems provide sophisticated mathematical capabilities, it is not suited as an environment for developing large applications such as XLOOPS, as it was not primarily intended as a programming language and only offers limited support for modern software engineering. For example, the only structured data type in **Maple** is the list, the distinction between local and global variables is not consistently enforced by all language constructs, and tools such as debuggers are very rudimentary.
- Different versions of **Maple** are in places incompatible with respect to the language. This leads to XLOOPS having to provide two versions of all program parts written in **Maple** one for **Maple** V Release 1 and one for Release 3. With Release 4 and later releases XLOOPS cannot currently be run. Not only does the parallel development of multiple program versions require higher maintenance effort, but it is also impractical for the user to require the installation of a specific version of **Maple** just to run XLOOPS.

- **Maple** is a commercial system, which prevents XLOOPS from getting a wide reach of distribution, especially in academic institutions.
- The communication between different program parts is difficult because **Maple** has insufficient support for embedding programs written in other languages, it requires higher efforts for the conversion of data and, in some cases, for the multiple storing of redundant information (for example, the information about graph topologies has to be duplicated for the calculation and for the graphical interface).
- In general, it is more difficult to maintain and develop a heterogenous program package. For the programmer it is necessary to become acquainted with three different environments (**Maple**, Tcl/Tk, and C).

The listed drawbacks and the observation of bugs related to the internal structure of XLOOPS led to the conclusion that the program had reached a state in which further development was almost impossible (approx. 15000 lines of badly documented source code, in some places very obscure programming techniques like identifiers whose meaning depends on capitalization). Thus, the decision was made to rewrite XLOOPS from scratch, putting it on more solid grounds.

In particular, it was decided that the new version of XLOOPS should be written in one uniform programming language. As the essential part of XLOOPS are the analytical calculations, traditional computer algebra systems (**Maple**, **Mathematica** [10], **Reduce** [11] and **MuPAD** [12]) were envisaged at first. But all these share more or less the same deficiencies as **Maple**, especially the low suitability of the built-in language for the development of large systems. It was therefore decided to use an established programming language as the foundation and extend it by the required algebraic capabilities. During the development of XLOOPS it was noted that only a small subset of the mathematical functions provided by **Maple** are actually needed. These are:

- complex arithmetic with arbitrary precision;
- simple manipulation of symbolic expressions, like expansion, collection, substitution of variables;
- simplification of rational functions;
- symbolic differentiation;
- special functions like polylogarithms and the Gamma function;
- Laurent series expansion of expressions containing these functions;
- solving systems of linear equations for the analysis of the tensor structure of the integrals;
- handling expressions containing elements of some particular non-commutative algebras such as Clifford and $SU(3)$ Lie algebras;
- numeric integration.

Features that are particularly *not* required are:

- symbolic integration since only few master integrals need to be really integrated and they will be done by hand;
- calculation of limits;
- treatment of domains and assumptions about the range of values of variables.

Due to the following reasons, C++ has been chosen as the programming language:

- C++ is officially standardized [13], so fewer complications are expected from the future development of the language.
- C++ allows to write down symbolic expressions in their natural mathematical notation by means of operator overloading (e.g. `4*a+b` instead of something like `add(mul(4,a),b)`).
- C++ is available for virtually all computer platforms. In particular, there are free compilers for the Unix systems predominant in the academic area.
- There is a large assortment of development tools available like powerful source-level debuggers and systems for version control and documentation.
- There is also a large number of existing libraries, especially for arbitrary precision arithmetics.
- As a compiled language, C++ is also suitable for numeric integration.

Based on C++ we developed a system called "GiNaC"¹ that is primarily aimed at the re-implementation of XLOOPS, but is also suited for the development of other systems that integrate algebraic and numeric calculations with user interfaces for methods of data acquisition.

3 GiNaC: A New Programming Environment For XLOOPS

GiNaC [1] is a C++ library for handling symbolic mathematical expressions². Some of the features of the library are

- complex arithmetic with arbitrary precision, based on the CLN library by Bruno Haible [14];
- manipulation of symbolic expressions;
- normalization of rational functions;
- matrices and systems of linear equations;
- numerous special functions (trigonometric and hyperbolic functions, exponential functions, logarithms, Gamma und polygamma functions);
- symbolic differentiation;

¹an acronym for "GiNaC is not a Computer Algebra System"

²<http://www.ginac.de>

numeric	real and complex numbers (integers like 42, exact fractions like $\frac{2}{3}$ and floating point numbers like 7.319), based on CLN
symbol	algebraic symbols
constant	constants like π which are treated similarly to symbols but also have a predefined numerical value
add	sums of the form $\sum_{i=1}^n c_i x_i$, for $c_i \in \mathbb{C}$ and arbitrary non-numeric expressions x_i
mul	product of the form $\prod_{i=1}^n x_i^{c_i}$, for $c_i \in \mathbb{C}$ and arbitrary non-numeric expressions x_i
power	arbitrary expressions of the form x^y
pseries	compact representation of Laurent and Taylor series (only contains the series coefficients, the expansion variable, and the expansion point)
function	mathematical functions like $\sin()$ und $\cos()$, where the individual functions are not implemented as subclasses of function but are distinguished by a function index
lst	lists of expressions
matrix	matrices
relational	equations and unequations

Table 1: The most important GiNaC classes

- series expansion of functions (Taylor and Laurent series);
- Clifford and SU(3) color algebras (this is a recently added feature in GiNaC, making it especially suited for applications in particle physics);
- it is Open Source, licensed under the GNU General Public License. This means that it is not only freely available but also free in the sense that users have access to the sources and are allowed to modify or extend the library and to redistribute the modified version. This is in sharp contrast to most other CAS that place heavy restrictions on their legal use.

GiNaC is designed in an object-oriented fashion. The central class of GiNaC is the class **ex** that stores a symbolic expression. Strictly speaking, **ex** only represents a "smart" pointer to the real expression which is stored as a tree whose nodes are objects subclassed from the abstract base class **basic**. The operators $+$, $-$, $*$ und $/$ are overloaded to simplify the creation of expressions in the program code.

Table 1 gives an overview of the most important subclasses of **basic**. The classes can be categorized into the "atomic" classes **numeric**, **symbol** and **constant** which are at the leaves of an expression tree,

and the container classes (all others) which themselves contain expressions. The representation of sums and products with numeric coefficients and powers was chosen for reasons of efficiency (see [15] and [16]).

A detailed description of the internal functionality of the basic classes and methods of GiNaC is given in [1] and [16].

4 One-Loop One-, Two-, And Three-Point Integrals

In this section we present our algorithm for tensor reduction, which was introduced by Collins [4] and further developed by Kreimer [2, 3]. A similar approach was used in a subsequent paper by Ghinculov and Yao [5]. The advantages of this approach compared to other procedures like the Passarino-Veltman procedure, especially with regard to future two-loop applications, are outlined in [16].

To regularize UV-divergences, we are working in $D = 4 - 2\epsilon$ dimensional space-time. The main idea is to split the space of integration in a parallel and an orthogonal space. We define the parallel space to be the linear span of the n -external momenta q_{i_μ} ($i = 1, \dots, n$) involved in the integrand. This parallel space has a finite dimension $J \leq D$. The remaining $D - J$ dimensions span an orthogonal complement of the parallel space called the orthogonal space [2].

Once an explicit configuration of external momenta is chosen, the dimension of the parallel space J is known, and the scalar products are written explicitly in terms of the components of the external momenta

$$\begin{aligned} l^2 &= l_0^2 - l_1^2 - \dots - l_{J-1}^2 - l_\perp^2, \\ l \cdot q_i &= l_0 q_{i_0} - l_1 q_{i_1} - \dots - l_{J-1} q_{i_{J-1}}. \end{aligned} \quad (1)$$

A general one-loop integral can be written as

$$\int d^D l F(l^2, \{l \cdot q_i\}) = \frac{2\pi^{\frac{D-J}{2}}}{\Gamma((D-J)/2)} \int_{-\infty}^{\infty} dl_0 \dots \int_{-\infty}^{\infty} dl_{J-1} \int_0^{\infty} dl_\perp l_\perp^{D-J-1} F(l_\perp, l_0, \dots, l_{J-1}). \quad (2)$$

In the usual Passarino-Veltman approach, to keep an explicitly covariant form, one expands tensor integrals of the type

$$T_{\mu_1 \dots \mu_N}^{(n)} = \int d^D l \frac{l_{\mu_1} \dots l_{\mu_N}}{\prod_{i=1}^n ((l + q_i)^2 - m_i^2 + i\rho)} \quad (3)$$

in a basis of Lorentz tensors constructed from the metric tensor $g_{\mu\nu}$ and the external momenta q_{i_μ} . Since $T_{\mu_1 \dots \mu_N}^{(n)}$ is symmetric, one can re-group indices belonging to the parallel space and to the orthogonal space together

$$T_{\mu_1 \dots \mu_N}^{(n)} = T_{\mu_J \dots \mu_D}^{(n)} \underbrace{\overbrace{0 \dots 0}^{p_0} \overbrace{1 \dots 1}^{p_1} \dots \overbrace{J-1 \dots J-1}^{p_{J-1}}}_{p_\parallel} \quad (4)$$

where p_0, p_1, \dots denote the numbers of indices for the 0-, 1-, \dots components. The indices μ_J, \dots, μ_D in the orthogonal space have to result in a symmetric combination of metric tensors in the orthogonal space $g_{\mu\nu}^\perp$

$$T_{\mu_1 \dots \mu_N}^{(n)} = \frac{(-1)^{\frac{D-J}{2}}}{K} \left(g_{\mu_J \mu_{J+1}}^\perp \cdots g_{\mu_{D-1} \mu_D}^\perp \right)_{\text{symm.}} T_{(p_0 \dots p_{J-1} p_\perp)}^{(n)}. \quad (5)$$

with $N = p_0 + \dots + p_{J-1} + p_\perp$. The normalization can be derived by looking at contractions with $g_{\mu\nu}^\perp$ and observing that

$$g_{\mu_J \mu_{J+1}}^\perp g^{\perp \mu_J \mu_{J+1}} = D - J. \quad (6)$$

One obtains

$$K = \prod_{i=J}^{\frac{D-J-2}{2}} (D - J + 2i). \quad (7)$$

The coefficients needed for the calculation of a specific component of a general one-loop tensor integral therefore have the form

$$T_{(p_0 \dots p_{J-1} p_\perp)}^{(n)} = \int d^D l \frac{l_0^{p_0} \cdots l_{J-1}^{p_{J-1}} l_\perp^{p_\perp}}{\prod_{i=1}^n \left((l + q_i)^2 - m_i^2 + i\rho \right)}. \quad (8)$$

In the next sections, we present an algorithm to calculate $T_{(p_0 \dots p_{J-1} p_\perp)}^{(n)}$ for the one-loop two- and three-point tensor integrals. In the rest of this paper, we call $T_{(p_0 \dots p_{J-1} p_\perp)}^{(n)}$ tensor integrals.

4.1 One-Loop Two-Point Tensor Integrals

In this section, the algorithm for an automatic calculation of one-loop two-point tensor integrals is presented. We first consider the case where q^2 is timelike, $q^2 > 0$. Then one can choose a reference

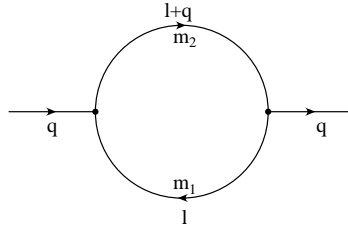


Figure 1: Notation for two-point functions

frame where $q_\mu = (q_0, 0, 0, 0)$. The general integral for a tensor Feynman diagram shown in Fig.1 has the form

$$I_{t_1 t_2}^{(2)ij}(q^2) = \int d^D l \frac{l_0^i l_\perp^j}{P_1^{t_1} P_2^{t_2}} \quad (9)$$

with

$$\begin{aligned} P_1 &= (l_0 + q_0)^2 - l_\perp^2 - m_1^2 + i\rho, \\ P_2 &= l_0^2 - l_\perp^2 - m_2^2 + i\rho, \end{aligned} \quad (10)$$

where l_0 and l_\perp span the parallel and orthogonal subspaces respectively and the integral vanishes unless j is even. In the spacelike and lightlike cases where $q^2 < 0$ or $q^2 = 0$ one can choose a reference frame where $q_\mu = (0, q_{01}, 0, 0)$ or $q_\mu = (q_{01}, q_{01}, 0, 0)$ respectively and the integral space can be split into a two-dimensional parallel and a $D - 2$ dimensional orthogonal subspace. We will consider these cases later in section 4.2.3. A genuine one-loop integral has $t_1 = t_2 = 1$, but the more general case is needed in the case of the reduction of integrals with more than one loop.

The strategy now is to express $I_{t_1 t_2}^{(2)ij}(q^2)$ as a polynomial of simpler integrals. It turns out that the usual scalar one- and two-point integrals A_0 and B_0 (in Passarino-Veltman notation [18, 19]) are sufficient. This expansion is always possible except for some special cases that we will consider later.

Firstly consider the general case where $q_0 \neq 0$. We express the numerator of the integral in Eq.(9) as a function of P_1 and P_2 . From Eq.(10) we get:

$$\begin{aligned} l_0 &\rightarrow l_0(P_1, P_2, q_0, m_1, m_2) = \frac{1}{2q_0}(P_1 - P_2 - C_1), \\ l_\perp^2 &\rightarrow l_\perp^2(P_1, P_2, q_0, m_1, m_2) = l_0^2 - P_2 + C_2, \\ C_1 &= q_0^2 - m_1^2 + m_2^2, \\ C_2 &= -m_2^2 + i\rho. \end{aligned} \quad (11)$$

Inserting Eq.(11) into Eq.(9) and expanding the numerator of the integrand, one obtains

$$I_{t_1 t_2}^{(2)ij}(q^2) = \sum_{n,m} C_{nm} \int d^D l P_1^{n-t_1} P_2^{m-t_2} \quad (12)$$

with C_{nm} being simple functions of q_0, m_1, m_2 , and

$$\int d^D l P_1^{n-t_1} P_2^{m-t_2} = \begin{cases} 0 & \text{if } n - t_1 \geq 0 \text{ and } m - t_2 \geq 0, \\ \int d^D l \frac{1}{P_1^{t_1-n} P_2^{t_2-m}} & \text{if } n - t_1 < 0 \text{ and } m - t_2 < 0, \\ \int d^D l \frac{P_1^{n-t_1}}{P_2^{t_2-m}} & \text{if } n - t_1 \geq 0 \text{ and } m - t_2 < 0, \\ \int d^D l \frac{P_2^{m-t_2}}{P_1^{t_1-n}} & \text{if } n - t_1 < 0 \text{ and } m - t_2 \geq 0. \end{cases} \quad (13)$$

We see that the second case actually corresponds to a scalar two-point function. For the last two cases, from Eq.(10) one can insert

$$P_1 = P_1(P_2, l_0, q_0) = 2l_0 q_0 + P_2 + C_1, \quad (14)$$

or

$$P_2 = P_2(P_1, l_0, q_0) = P_1 - 2l_0q_0 - C_1 \quad (15)$$

and expand the numerator of the integrands in Eq.(13). Then Eq.(10) can be completely reduced to the one-point functions

$$I_t^{(1)i} = \int d^D l \frac{(l^2)^{\frac{i}{2}}}{(l^2 - m^2 + i\rho)^t}. \quad (16)$$

4.1.1 The case $q_0 = 0$

If $q_0 = 0$, Eq.(13) must be rewritten using

$$\begin{aligned} P_1 &= l_0^2 - l_\perp^2 - m_1^2 + i\rho, \\ P_2 &= l_0^2 - l_\perp^2 - m_2^2 + i\rho. \end{aligned} \quad (17)$$

If $m_1 = m_2$ then $P_1 = P_2$ and $I_{t_1 t_2}^{(2)ij}(q^2)$ has the simple form

$$I_{t_1 t_2}^{(2)ij}(q^2) = \int d^D l \frac{l_0^i l_\perp^j}{P_1^{t_1+t_2}} \quad (18)$$

that is actually the one-loop one-point function.

If $m_1 \neq m_2$, one performs partial fraction decomposition and finds

$$\begin{aligned} I_{t_1 t_2}^{(2)ij}(q^2) &= \int d^D l \frac{l_0^i l_\perp^j}{P_1(m_1^2)^{t_1} P_2(m_2^2)^{t_2}} \\ &= \frac{1}{(t_1 - 1)!} \frac{d^{t_1-1}}{d(m_1^2)^{t_1-1}} \left(\frac{1}{(m_1^2 - m_2^2)^{t_2}} \int d^D l \frac{l_0^i l_\perp^j}{P_1(m_1^2)} \right) + \\ &\quad \frac{1}{(t_2 - 1)!} \frac{d^{t_2-1}}{d(m_2^2)^{t_2-1}} \left(\frac{1}{(m_2^2 - m_1^2)^{t_1}} \int d^D l \frac{l_0^i l_\perp^j}{P_2(m_2^2)} \right), \end{aligned} \quad (19)$$

which is a combination of one-loop one-point functions.

4.2 One-Loop Three-Point Tensor Functions

The notation we use for one-loop three-point functions is shown in Fig.2. We are working in the frame of reference where the external momentum configuration is $q_{1\mu} = (q_{10}, 0, 0, 0)$, $q_{2\mu} = (q_{20}, q_{21}, 0, 0)$. The parallel space is now two-dimensional and the general form of the one-loop three-point tensor function is

$$I_{t_1 t_2 t_3}^{(3)ijk} = \int d^D l \frac{l_0^i l_1^j l_\perp^k}{P_1^{t_1} P_2^{t_2} P_3^{t_3}} \quad (20)$$

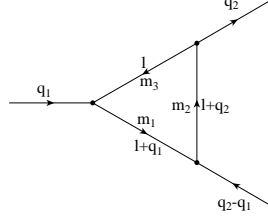


Figure 2: Notation for three-point functions

with

$$\begin{aligned}
P_1 &= l_0^2 - l_1^2 - l_\perp^2 + 2l_0 q_{10} + q_{10}^2 - m_1^2 + i\rho, \\
P_2 &= l_0^2 - l_1^2 - l_\perp^2 + 2l_0 q_{20} - 2l_1 q_{21} + q_{20}^2 - q_{21}^2 - m_2^2 + i\rho, \\
P_3 &= l_0^2 - l_1^2 - l_\perp^2 - m_3^2 + i\rho
\end{aligned} \tag{21}$$

where $\{l_0, l_1\}$ and l_\perp span the parallel and orthogonal subspaces, respectively.

4.2.1 The general case

Firstly we consider the case where $q_{10} \neq 0$ and $q_{21} \neq 0$. Then we are always able to express l_0, l_1, l_\perp in terms of P_1, P_2, P_3 :

$$\begin{aligned}
l_0 &= \frac{1}{2q_{10}}(P_1 - P_3 - c_{00}), \\
l_1 &= \frac{1}{2q_{10}q_{21}} [q_{20}(P_1 - P_3 - (q_{10}^2 - m_1^2 + m_3^2)) + q_{10}(P_3 - P_2) + q_{10}(m_3^2 - m_2^2 + q_{20}^2 - q_{21}^2)] \\
&= \frac{1}{2q_{21}} [c_{10}l_0 + (P_3 - P_2) + c_{11}], \\
l_\perp^2 &= l_0^2 - l_1^2 - P_3 - c_{20}
\end{aligned} \tag{22}$$

with

$$\begin{aligned}
c_{00} &= q_{10}^2 - m_1^2 + m_3^2, \\
c_{10} &= 2q_{20}, \\
c_{11} &= m_3^2 - m_2^2 + q_{20}^2 - q_{21}^2, \\
c_{20} &= m_3^2 - i\rho.
\end{aligned} \tag{23}$$

Again, as in the case of two-points functions, one substitutes Eq.(22) into Eq.(20) and obtains a combination of scalar three-point functions

$$I_{t_1 t_2 t_3}^{(3)ijk} = \sum_{m,n,r}^{i+j+k} C_{mnr} \int d^D l P_1^{(m-t_1)} P_2^{(n-t_2)} P_3^{(r-t_3)} \tag{24}$$

with C_{mnr} being simple functions of masses and components of external momenta. More explicitly, the integrand on the right hand side of Eq.(24) contains terms like

$$\frac{1}{P_1^{n_1} P_2^{n_2} P_3^{n_3}}, \quad \frac{P_i^{n_i} P_j^{n_j}}{P_k^{n_k}}, \quad \text{and} \quad \frac{P_i^{n_i}}{P_j^{n_j} P_k^{n_k}} \quad (25)$$

with $i \neq j \neq k$ and positive n_i . Other possible combinations lead to a vanishing integral.

The first group of terms can be obtained from derivatives of scalar three-point functions. For the last two cases, using Eq.(21) one can expand the numerator in terms of propagators P_{n_j} in the denominator. This expansion is always possible and reduces the second group of terms to one-point functions. Similarly the third group of terms can be reduced to one-loop two-point functions. Note that in this step we meet two kinds of one-loop two-point functions. The first kind is the one-loop two-point function with one parallel dimension that was already found in the previous section. The second kind is the one-loop two-point function with two parallel dimensions of the internal momentum l that is not trivial and will be given in the section 4.2.3 as a separate case.

4.2.2 The case $q_{10} q_{21} = 0$

In this case the expansions in Eq.(22) cannot be used. However, Eq.(20) can be reduced by partial fraction decomposition. First consider the case $q_{10} = 0$ with arbitrary values of q_{21} .

4.2.2.1 The case $q_{10} = 0$

Eq.(21) simplifies to

$$\begin{aligned} P_1 &= l_0^2 - l_1^2 - l_\perp^2 - m_1^2 + i\rho, \\ P_2 &= l_0^2 - l_1^2 - l_\perp^2 + 2l_0 q_{20} - 2l_1 q_{21} + q_{20}^2 - q_{21}^2 - m_2^2 + i\rho, \\ P_3 &= l_0^2 - l_1^2 - l_\perp^2 - m_3^2 + i\rho. \end{aligned} \quad (26)$$

If $m_1 \neq m_3$, partial fraction decomposition leads to a separation into terms which contain only two propagators

$$I_{t_1 t_2 t_3}^{(3)ijk} = \prod_{f=1}^3 \left(\frac{1}{(t_f - 1)!} \frac{d^{t_f-1}}{d(m_f^2)^{t_f-1}} \right) \left\{ \frac{1}{m_1^2 - m_3^2} \left(\int d^D l \frac{l_0^i l_1^j l_\perp^k}{P_1(m_1^2) P_2(m_2^2)} - \int d^D l \frac{l_0^i l_1^j l_\perp^k}{P_2(m_2^2) P_3(m_3^2)} \right) \right\}. \quad (27)$$

If $m_1 = m_3$, two propagators are equal and it is sufficient to calculate

$$I_{t_1 t_2 t_3}^{(3)ijk} = \int d^D l \frac{l_0^i l_1^j l_\perp^k}{P_1^{t_1+t_3}(m_1^2) P_2^{t_2}(m_2^2)}. \quad (28)$$

The three-point integrals are then reduced to combinations of two-point integrals with two parallel dimensions. We will treat this class of two-point functions in section 4.2.3.

4.2.2.2 The case $q_{10} \neq 0$ and $q_{21} = 0$

In this special case, the integrals collapse from two parallel dimensions to one and Eq.(21) reads

$$\begin{aligned} P_1 &= (l_0 + q_{10})^2 - l_\perp'^2 - m_1^2 + i\rho, \\ P_2 &= (l_0 + q_{20})^2 - l_\perp'^2 - m_2^2 + i\rho, \\ P_3 &= l_0^2 - l_\perp'^2 - m_3^2 + i\rho \end{aligned} \quad (29)$$

where the components l_1 and l_\perp can be combined into l_\perp' to form a new $D - 1$ dimensional orthogonal subspace with

$$l_\perp'^2 = l_1^2 + l_\perp^2. \quad (30)$$

Then the integral can be reduced to a simpler integral with only one parallel dimension

$$I_{t_1 t_2 t_3}^{(3)i,j+k} = \int d^D l \frac{l_0^i l_\perp^{j+k}}{P_1^{t_1} P_2^{t_2} P_3^{t_3}}. \quad (31)$$

Again, one can use the same procedure as in the general case

$$\begin{aligned} l_0 &\rightarrow l_0(P_1, P_3, m_i, q_{10}) = \frac{1}{2q_{10}}(P_1 - P_3 + m_1^2 - m_3^2 - q_{10}^2), \\ l_\perp &\rightarrow l_\perp(P_1, P_3, m_i, q_{10}) = l_0^2 - P_3 - m_3^2 + i\rho \end{aligned} \quad (32)$$

that reduces Eq.(31) to a form similar to Eq.(24).

4.2.3 The two-point integral with two parallel dimensions J_2^{ijk}

In the preceding section we have shown that general one-loop three-point tensor functions can be reduced to the usual scalar integrals and one new two-point function corresponding to a tensor component in a two-dimensional parallel space. Explicitly, this integral reads

$$\begin{aligned} J_2^{ijk} &= \int d^D l \frac{l_0^i l_1^j l_\perp^k}{[(l_0 + q_{10})^2 - l_1^2 - l_\perp^2 - m_1^2 + i\rho]^{t_1} [(l_0 + q_{20})^2 - (l_1 + q_{21})^2 - l_\perp^2 - m_2^2 + i\rho]^{t_2}} \\ &= \int d^D l \frac{(l_0 - q_{10})^i l_1^j l_\perp^k}{[l_0^2 - l_1^2 - l_\perp^2 - m_1^2 + i\rho]^{t_1} [(l_0 + Q_0)^2 - (l_1 + Q_1)^2 - l_\perp^2 - m_2^2 + i\rho]^{t_2}} \end{aligned} \quad (33)$$

with $Q_0 = q_{20} - q_{10}$, $Q_1 = q_{21}$; $Q_\mu = q_{2\mu} - q_{1\mu}$. If $Q_1 = 0$, this integral reduces to the one-loop two-point function in one-dimensional parallel space as found in the previous section. If, on the other hand, $Q_1 \neq 0$, one can always find a Lorentz boost which transforms into a reference frame where the transformed 4-momentum Q'_μ has either only one non-zero component (Q'_0 or Q'_1 if Q is timelike or spacelike) or where $Q'_0 = Q'_1$ if Q is lightlike. The loop momentum has to be boosted accordingly which, however, modifies only the numerator of the integrand in Eq.(33). Explicitly, consider the boost

$$\begin{pmatrix} l_0 \\ l_1 \end{pmatrix} = \begin{pmatrix} \gamma & \gamma\beta \\ \gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} l'_0 \\ l'_1 \end{pmatrix}. \quad (34)$$

Then the three sub-cases are treated as follows.

4.2.3.1 The timelike case $Q_0^2 - Q_1^2 > 0$

In this case, under the transformation in Eq.(34) the integral will be reduced to one-loop two-point functions with a one-dimensional parallel space as found in the previous section:

$$J_2^{ijk} = \int d^D l' \frac{[\gamma(l'_0 + \beta l'_1) - q_1]^i [\gamma(\beta l'_0 + l'_1)]^j l'^k_{\perp}}{[l'^2_0 - l'^2_1 - l'^2_{\perp} - m_1^2 + i\rho]^{t_1} [(l'_0 + P)^2 - l'^2_1 - l'^2_{\perp} - m_2^2 + i\rho]^{t_2}} \quad (35)$$

with $P = \sqrt{Q_0^2 - Q_1^2}$, $\gamma = Q_0/P$ and $\beta = Q_1/Q_0$.

4.2.3.2 The spacelike case, $Q_0^2 - Q_1^2 < 0$

In this case the boost with $P = \sqrt{Q_1^2 - Q_0^2}$, $\gamma = Q_1/P$ and $\beta = Q_0/Q_1$ transforms to a reference frame in which the integral reads

$$J_2^{ijk} = \int d^D l' \frac{[\gamma(l'_0 + \beta l'_1) - q_1]^i [\gamma(\beta l'_0 + l'_1)]^j l'^k_{\perp}}{[l'^2_0 - l'^2_1 - l'^2_{\perp} - m_1^2 + i\rho]^{t_1} [l'^2_0 - (l'_1 + P)^2 - l'^2_{\perp} - m_2^2 + i\rho]^{t_2}} \quad (36)$$

The components l'_0 and l'_{\perp} can be combined to form a new $D - 1$ dimensional orthogonal subspace while l_1 spans the parallel subspace. Using the same procedure as in the previous sections one can reduce the integral completely to the scalar one- and two-point functions.

4.2.3.3 The lightlike case, $Q_0^2 - Q_1^2 = 0$

In this case the transformation in Eq.(34) is singular and the integral J_2^{ijk} becomes

$$J_2^{ijk} = \int d^D l \frac{(l_0 - q_1)^i l^j_1 l^k_{\perp}}{[l_0^2 - l_1^2 - l_{\perp}^2 - m_1^2 + i\rho]^{t_1} [(l_0 + Q_0)^2 - (l_1 + Q_0)^2 - l_{\perp}^2 - m_2^2 + i\rho]^{t_2}}. \quad (37)$$

By solving the system of equations

$$\begin{aligned} P_1 &= l_0^2 - l_1^2 - l_{\perp}^2 - m_1^2 + i\rho, \\ P_2 &= (l_0 + Q_0)^2 - (l_1 + Q_0)^2 - l_{\perp}^2 - m_2^2 + i\rho, \end{aligned} \quad (38)$$

one obtains

$$\begin{aligned} l_{\perp}^2 &= l_0^2 - l_1^2 - P_1 - m_1^2 + i\rho, \\ l_1 &= \frac{P_1 - P_2 + m_2^2 - m_1^2}{2Q_0} + l_0. \end{aligned} \quad (39)$$

Inserting Eq.(39) into Eq.(37) and expanding the numerator of the integrand, the integral will be reduced to scalar one-point functions and a simpler tensor integral

$$J_2^i = \int d^D l \frac{(l_0)^i}{[l_0^2 - l_1^2 - l_{\perp}^2 - m_1^2 + i\rho]^{t_1} [(l_0 + Q_0)^2 - (l_1 + Q_0)^2 - l_{\perp}^2 - m_2^2 + i\rho]^{t_2}}. \quad (40)$$

The explicit calculation of this integral is given in [17].

This completes the description of our algorithm for tensor reduction. We did not reproduce explicit expressions for the basic scalar integrals in this paper since these can be found in the literature [2, 19].

5 Conclusion

Due to the limitations of Maple and the internal structure of XLOOPS we decided to rewrite XLOOPS from scratch, based on GiNaC, a C++ library to replace Maple as an algebraic programming environment [1]. An efficient algorithm for one-loop one-, two- and three-point tensor reduction was also successfully implemented. At this stage of the project, a package for calculating one-loop one-, two- and three-point tensor integrals is available and can be downloaded from <http://wwwthep.physik.uni-mainz.de/~xloops>. Like GiNaC, it is distributed under the terms of the GNU General Public License.

As the next step, we plan to rewrite the module for two-loop one-, two- and three-point integrals and to completely recode XLOOPS in C++ using the GiNaC library, providing a package for doing particle physics in a homogenous C++ environment.

Acknowledgements

Part of this work is supported by the DFG-Forschungsproject "KO 1069/6-1" and the "Graduiertenkolleg Eichtheorien — Experimentelle Tests und theoretische Grundlagen" at the University of Mainz. The authors would like to thank Jürgen Körner, Dirk Kreimer and Hubert Spiesberger for many fruitful comments and corrections, and Lars Brücher, Alexander Frink and Richard Kreckel for inspiring discussions.

Appendix A

In this section we introduce the definitions for the one-loop integral functions in XLOOPS-GiNaC. The XLOOPS-GiNaC package provides the three GiNaC functions `OneLoop1Pt()`, `OneLoop2Pt()`, and `OneLoop3Pt()`, which can be used in algebraic expressions like any other predefined GiNaC function:

- The one-point function

$$\text{OneLoop1Pt}(\mathbf{i}, \mathbf{m}, \mathbf{t}, \rho) = I_t^{(1)i} = \int d^D l \frac{(l^2)^{\frac{i}{2}}}{[l^2 - m^2 + i\rho]^t}.$$

- The two-point function

$$\begin{aligned} \text{OneLoop2Pt}(\mathbf{i}, \mathbf{j}, \mathbf{q}, \mathbf{m}_1, \mathbf{m}_2, \mathbf{t}_1, \mathbf{t}_2, \rho) &= I_{t_1 t_2}^{(2)ij} \\ &= \int d^D l \frac{l_0^i l_\perp^j}{[(l_0 + q)^2 - l_\perp^2 - m_1^2 + i\rho]^{t_1} [l_0^2 - l_\perp^2 - m_2^2 + i\rho]^{t_2}}. \end{aligned}$$

- The three-point function

$$\begin{aligned} \text{OneLoop3Pt}(\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{q}_{10}, \mathbf{q}_{20}, \mathbf{q}_{21}, \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \rho) &= I_{t_1 t_2 t_3}^{(3)ijk} \\ &= \int d^D l \frac{l_0^i l_1^j l_\perp^k}{P_1^{t_1} P_2^{t_2} P_3^{t_3}} \end{aligned}$$

with

$$\begin{aligned} P_1 &= l_0^2 - l_1^2 - l_\perp^2 + 2 l_0 q_{10} + q_{10}^2 - m_1^2 + i\rho, \\ P_2 &= l_0^2 - l_1^2 - l_\perp^2 + 2 l_0 q_{20} - 2 l_1 q_{21} + q_{20}^2 - q_{21}^2 - m_2^2 + i\rho, \\ P_3 &= l_0^2 - l_1^2 - l_\perp^2 - m_3^2 + i\rho. \end{aligned}$$

As with any GiNaC function, the arguments and return values of the above functions are objects of type `ex` so the return values as well as input parameters can be any symbolic or numeric expression.

In order to illustrate the output of XLOOPS-GiNaC, we give one example program that calculates and prints out both analytical and numerical results of the UV-divergent and the finite terms of the integral `OneLoop2Pt(1, 0, q, m1, m2, 1, 1, rho)`

```

1  #include <iostream>
2  #include <ginac/ginac.h>
3  #include <xloops/xloops.h>
4  using namespace GiNaC;
5  using namespace xloops;
6
7  int main()
8  {
9      symbol q("q"), m1("m1"), m2("m2"), eps("eps"), rho("rho");
10     ex a = OneLoop2Pt(1, 0, q, m1, m2, 1, 1, rho);
11     a = a.series(eps == 0, 4);
12     ex a1 = a.coeff(eps, -1).subs(rho == 0);
13     ex a2 = a.coeff(eps, 0).subs(rho == 0);
14     cout << "Order eps^-1 is " << endl << a1.normal() << endl;
15     cout << "Order eps^0 is " << endl << a2.normal() << endl;
16     ex b1 = a.subs(rho==0).subs(m1==80).subs(m2==80).subs(q==100).evalf();
17     cout << "Numerical value up to order eps^2 is " << endl << b1 << endl;
18     return 0;
19 }
```

The output of this program reads

```

1  Order eps^-1 is
2  -1/2*I*Pi^2*q
3
4  Order eps^0 is
5  (-1/2*q^(-1)*m2^2-1/2*q+1/2*q^(-1)*m1^2)*(I*Pi^2*q^(-1)*((1/2*q-1/2*q^(-1)*(m1
6  ^2-m2^2))*R2ex1(-m2^2,-(1/2*q-1/2*q^(-1)*(m1^2-m2^2))^2)+R2ex1(-m1^2,-(1/2*q+1
7  /2*q^(-1)*(m1^2-m2^2))^2)*(1/2*q+1/2*q^(-1)*(m1^2-m2^2)))+q*(-I*Pi^2*log(Pi)*q
8  ^(-1)+Pi^(3/2)*(I*sqrt(Pi)*q^(-1)*(-2*log(2)-Euler)+2*I*log(2)*sqrt(Pi)*q^(-1)
9  +2*I*sqrt(Pi)*q^(-1))-Pi^3)+1/2*q^(-1)*(-I*Pi^2*m2^2*log(m2^2)-I*Pi^2*log(Pi)
10 *m2^2+Pi^2*(I-I*Euler)*m2^2)-1/2*q^(-1)*(Pi^2*(-I*m1^2*log(m1^2)-I*Euler*m1^2+
11 I*m1^2)-I*Pi^2*log(Pi)*m1^2)
12
13 Numerical value up to order eps^2 is
14 (-493.48022005446793098*I)*eps^(-1)+(5019.9161138633880865*I)+(-4.6074255521
15 943996428E-15-25944.085010687200793*I)*eps+Order(eps^2)
```

Appendix B

In order to illustrate the use of XLOOPS-GiNaC for calculating one-loop Feynman diagrams, we give one example program that is actually part of the automated regression tests of the package. It checks that the longitudinal part of the vacuum polarization in QED vanishes on the one-loop level as required by gauge invariance, i.e.

$$q_\mu \Pi^{\mu\nu}(q^2) = 0, \quad (41)$$

where

$$\Pi^{\mu\nu}(q^2) = -e^2 \int d^D l \operatorname{Tr} \frac{\gamma^\mu (\not{l} + \not{q} + m) \gamma^\nu (\not{l} + m)}{((l+q)^2 - m^2)(l^2 - m^2)}.$$

The general tensor structure of $\Pi^{\mu\nu}$ is

$$\Pi^{\mu\nu} = A g^{\mu\nu} + B \frac{q^\mu q^\nu}{q^2}$$

with functions A and B that, because of (41), satisfy

$$A + B = 0.$$

This expression is obtained by contracting $\Pi^{\mu\nu}$ with $\frac{q_\mu q_\nu}{q^2}$:

$$\begin{aligned} \Pi^{\mu\nu} \frac{q_\mu q_\nu}{q^2} &= A + B \\ &= -\frac{e^2}{q^2} \int d^D l \operatorname{Tr} \frac{\not{q}(\not{l} + \not{q} + m) \not{q}(\not{l} + m)}{((l+q)^2 - m^2)(l^2 - m^2)}. \end{aligned} \quad (42)$$

The following program verifies that this expression vanishes for the first three orders of the regularization parameter:

```

1  #include <ginac/ginac.h>
2  #include <xloops/xloops.h>
3  using namespace GiNaC;
4  using namespace xloops;
5
6  int main(void)
7  {
8      symbol D("D");
9      symbol l("l"), q("q"), m("m"), e("e");
10     symbol l0("l0"), lorth("lorth"), eps("eps"), rho("rho");
11
12     scalar_products sp;
13     sp.add(l, l, pow(l0, 2) - pow(lorth, 2));
14     sp.add(q, q, pow(q, 2));
15     sp.add(l, q, l0*q);
16
17     ex I = -pow(e, 2) / pow(q, 2)

```



```

18         * dirac_slash(q, D)
19         * (dirac_slash(l, D) + dirac_slash(q, D) + m * dirac_ONE())
20         * dirac_slash(q, D)
21         * (dirac_slash(l, D) + m * dirac_ONE());
22 I = dirac_trace(I).simplify_indexed(sp);
23
24 ex a;
25 for (int i=0; i<3; i++)
26     for (int j=0; j<3; j++) {
27         ex c = I.coeff(l0, i).coeff(lorth, j);
28         a += c * OneLoop2Pt(i, j, q, m, m, 1, 1, rho);
29     }
30 a = a.series(eps == 0, 4);
31
32 ex a1 = a.coeff(eps, -1).subs(rho == 0);
33 ex a2 = a.coeff(eps, 0).subs(rho == 0);
34 ex a3 = a.coeff(eps, 1).subs(rho == 0);
35
36 cout << "Order eps^-1 is " << a1.expand() << endl;
37 cout << "Order eps^0 is " << a2.expand() << endl;
38 cout << "Order eps^1 is " << a3.expand() << endl;
39
40 return 0;
41 }

```

Line	Explanation
1-4	Include the header files for the GiNaC and XLOOPS libraries.
8-10	Declaration of all appearing symbols. These are the spacetime dimension D , the loop and external momenta l and q , the mass m , the parallel and orthogonal space loop components l_0 and l_\perp , the dimensional regularization parameter ε , and the infinitesimal imaginary part of the propagator ρ . With GiNaC it is necessary to specify the name used for printing expressions because C++ does not provide the names of variables at run-time.
12-15	The possible scalar products of the momenta are expressed in terms of q , l_0 and l_\perp and are registered in a <code>scalar_products</code> object which is later passed to <code>simplify_indexed()</code> : $l \cdot l = l_0^2 - l_\perp^2$, $q \cdot q = q^2$, $l \cdot q = l_0 q$. <code>pow()</code> is used for exponentiation because the C++ operator <code>^</code> has the wrong precedence in relation to the operator <code>*</code> .
17-21	The numerator of the integrand of Eq.(42) is constructed using Clifford algebra objects in a nearly 1-to-1 translation of the right-hand side of that equation. Note that GiNaC does not require the use of a special operator for non-commutative products here.
22	The trace is taken and the resulting expression which contains metric tensors is simplified, inserting the scalar products defined above (the result for \mathbf{I} is $-4e^2(l_0^2 + l_\perp^2 + ql_0 + m^2)$). Expressions are usually manipulated in the C++ oriented notation <i>expression.function(parameters)</i> , but the functional notation <i>function(expression, parameters)</i> is also available, as shown.
24-29	The integral is now expressed in terms of the basic <code>OneLoop2Pt()</code> integral functions by assembling the coefficients of all powers of l_0 and l_\perp .
30	To get the UV divergence as well as the finite part, one needs to take the series expansion of the integral at the pole $\varepsilon = 0$.
32-34	The coefficients of the series for the orders ε^{-1} , ε^0 and ε^1 are extracted and the limit $\rho \rightarrow 0$ is taken by calling the function <code>subs()</code> .
36-38	The three coefficients are simplified with <code>expand()</code> and printed to the standard output stream. When run, this program will output all three coefficients as 0.

BIBLIOGRAPHY

1. C. Bauer, A. Frink, R. Kreckel, *Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language*, arXiv: cs.SC/0004015, to appear in J. Symb. Comput.
2. D. Kreimer, *Dimensional Regularization in the Standard Model*, Dissertation, Universität Mainz 1992.
3. D. Kreimer, *Tensor Integrals for two-loop Standard Model Calculations*, Univ. of Tasmania preprint UTAS-PHYS-93-40, hep-ph/9312223
4. J. Collins, *Renormalization*, Cambridge Univ. Press, 1984
5. A. Ghinculov, Y.-P. Yao, Nucl. Phys. B 516:385-401, 1998, hep-ph/9702266
6. L. Brücher, J. Franzkowski, D. Kreimer,
A new Method for Computing One-Loop Integrals, Comput. Phys. Commun. 85 (1995) 153;
Oneloop 2.0 – A Program Package calculating One-Loop Integrals, Comput. Phys. Commun. 107 (1997) 281-292;
 L. Brücher, J. Franzkowski, A. Frink, D. Kreimer, *Introduction to xloops*, Nucl. Instrum. Meth. A389 (1997) 323-342.
7. T. Hahn, *Automatic loop calculations with FeynArts, FormCalc and LoopTools*, Nucl. Phys. Proc. Suppl. 89:231-236, 2000, hep-ph/0005029
8. B. W. Char et al., *Maple V language reference manual*, Springer, New York, 1991.
9. J. Ousterhout, *Tcl And The Tk Toolkit*, Addison-Wesley, Redwood City, Calif., 1994.
10. S. Wolfram, *Mathematica: a system for doing mathematics by computer, 2nd Edition*, Addison-Wesley, Redwood City, Calif., 1991.
11. A. C. Hearn, *Reduce User's Manual Version 3.5*, RAND Publication, Santa Monica, 1993.
12. W. Oevel, F. Postel, G. Rüschler, S. Wehmeier, *Das MuPAD Tutorium*, Springer, Berlin, 1999.
13. American National Standards Institute, ISO/IEC 14882-1998(E), *Programming Languages — C++*, 1998.
14. B. Haible, CLN, *a Class Library for Numbers*, <http://clisp.cons.org/~haible/packages-cln.html>
15. K. O. Geddes, S. R. Czapor, G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publishers, Boston, 1992.
16. A. Frink, *Computer-algebraische und analytische Methoden zur Berechnung von Vertexfunktionen im Standardmodell*, Dissertation, Universität Mainz, 2000.
17. L. Brücher, *Automatische Berechnung von Strahlungskorrekturen in Perturbativen Quantenfeldtheorien*, Dissertation, Universität Mainz, 1997.
18. G. Passarino, M. Veltman, Nucl. Phys. B160 (1979) 151.
19. G. 't Hooft, M. Veltman, Nucl. Phys. B153 (1979) 365-401.
 A. Davydychev, R. Delbourgo, J. Math. Phys. 39, (1998), 4299, hep-th/9709216.